tkz-elements 3D

Drawing solids with tkz-elements

Alain Matthes

Version 6.12c — 2026/11/26 (experimental)

Abstract

This document describes an extension of `tkz-elements` for drawings in space: polyhedra, spheres, cylinders, cones, plane faces, curves and graphs, with their visible and hidden parts. The spirit is the one of `tkz-elements`: the figure is computed in Lua, the points of the drawing are stored in the table `z`, and the drawing is done with `tkz-euclide`. It is not a new drawing language.

Contents

1 Getting started: a 3D figure step by step **3**
- 1.1 The idea . . . 3
- 1.2 Seven steps . . . 3
- 1.3 A first example: a cube and one of its diagonals . . . 3
- 1.4 Two traps . . . 4

2 Loading the extension **4**

3 Points and vectors of space **5**

4 The camera **5**
- 4.1 Changing the point of view . . . 6

5 Objects **7**

6 The scene: visible and hidden parts **8**

7 Drawing **9**

8 The ellipsoid **10**

9 Transformations and graphs **12**

10 Space geometry in the classroom **14**
- 10.1 Section of a cube by the plane (IJK) . . . 14
- 10.2 Two secant planes and their line of intersection . . . 15
- 10.3 A plane and a sphere: the circle of intersection . . . 16
- 10.4 Roof theorem in a tetrahedron . . . 17
- 10.5 Distance from a point to a plane: O and the plane (ABC) . . . 18

11 Gallery **19**

12 Limits of the present version **19**

1 Getting started: a 3D figure step by step

1.1 The idea

A 3D figure is *described in space and drawn in the plane*. You give the points and the solids in space; a *camera* projects them onto the page; the *scene* sorts each line into a visible part and a hidden part. The drawing itself is done, as in 2D, with `tkz-euclide`.

Three tables are used:

- `z3`: the points of space (class `point3`);
- `z`: the points of the drawing (class `point`, as usual);
- the *groups* of lines of the scene (for example `"cube"` or `"diag"`), each one split into visible and hidden pieces.

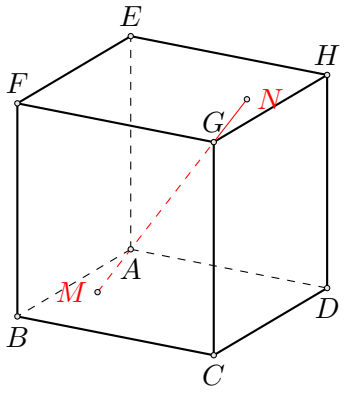
1.2 Seven steps

1. **Choose the point of view:** `cam = camera{azimuth = 30, elevation = 20}`.
2. **Create the scene:** `S = scene3(cam)`.
3. **Place the points of space:** `z3.O = point3(0, 0, 0)`.
4. **Build the solids and add them to the scene**, so that they can hide what is behind them: `CU = S:add(cube(z3.O, 4))`.
5. **Say what is to be drawn, group by group:** `S:polyhedron("cube", CU)`, `S:segment("diag", M, N)`, `S:circle(...)`… Visible and hidden parts are computed here.
6. **Finish and prepare the labels:** `S:export()`, then `cam:to_z{...}` (or `CU:vertices_to_z`) to put in `z` the points to be named.
7. **Draw:** in the `tikzpicture`, `\tkzGetNodes`, then for each group `\tkzDrawVisible[...]{group}` and `\tkzDrawHidden[...]{group}`, and the points and labels with `tkz-euclide`.

1.3 A first example: a cube and one of its diagonals

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 30, elevation = 20}
  local S = scene3(cam)
  z3.O = point3(0, 0, 0)
  local CU = S:add(cube(z3.O, 3))
  S:polyhedron("cube", CU)
  z3.A, z3.G = CU.vertices[1], CU.vertices[7]
  z3.M = z3.A - 0.4 * (z3.G - z3.A)
  z3.N = z3.G + 0.4 * (z3.G - z3.A)
  S:segment("diag", z3.M, z3.N)
  S:export()
  CU:vertices_to_z(cam, {"A","B","C","D","E","F","G","H"})
  cam:to_z{M = z3.M, N = z3.N}
}
\begin{tikzpicture}
  \tkzGetNodes
```

```
  \tkzDrawVisible[thick]{cube}  \tkzDrawHidden{cube}
  \tkzDrawVisible[red]{diag}    \tkzDrawHidden[red, dashed]{diag}
  \tkzDrawPoints(A,B,C,D,E,F,G,H,M,N)
  \tkzLabelPoints(A,B,C,D)
  \tkzLabelPoints[above](E,F,G,H)
  \tkzLabelPoints[red, left](M)
  \tkzLabelPoints[red, right](N)
\end{tikzpicture}
```



The line (MN) contains the diagonal $[AG]$ of the cube; A is the farthest vertex and G the nearest one. `\tkzDrawVisible` draws the visible pieces of a group, `\tkzDrawHidden` its hidden pieces: here $[MA]$ is hidden by the cube, $[AG]$ is inside the cube, only $[GN]$ is visible. A group may have no piece of one kind; the macro then draws nothing.

The cube is opaque: its three back edges are dashed. With `cube(z3.0, 3, false)` the cube becomes a *wireframe*: its back edges are still dashed, but it hides nothing.

1.4 Two traps

- No Lua comment `--` inside `\directlua`: the line ends are removed and the comment would swallow the rest of the code. Use `%`.
- Avoid names of points ending with `p` (`lp`, `Ap`…): `tkz-elements` turns them into `l'`, `A'`.

2 Loading the extension

The extension is loaded by an option of `tkz-elements`:

```
\usepackage[3d]{tkz-elements}
```

Without this option nothing is loaded: the 2D part of `tkz-elements` is not modified. The option loads the package `tkz-elements-3d`, made of the Lua file `tkz_elements_3d.lua` and of the drawing macros (`\tkzDrawVisible`, `\tkzDrawHidden`…). Another package can also load it with `\RequirePackage{tkz-elements-3d}`.

This module is *experimental*: names and options may still change.

The source of this document is given with the package (`doc/latex-3d/tkz-elements-3d.tex`): every figure is built from the code shown above it.

3 Points and vectors of space

The class `point3` represents both the points and the vectors of space. The points are stored in the table `z3` (the analogue of `z`).

Syntax	Result
`point3(x, y, z)`	the point (or vector) (x, y, z)
`P + Q, P - Q, -P`	sum, difference, opposite
`k * P, P * k, P / k`	product by a number
`P:dot(Q)`	scalar product
`P:cross(Q)`	cross product
`P:norm(), P:unit()`	norm, unit vector
`midpoint3(A, B)`	midpoint
`barycenter3({A, B, C})`	isobarycenter

4 The camera

The camera defines the projection of space onto the page. Three projections are available.

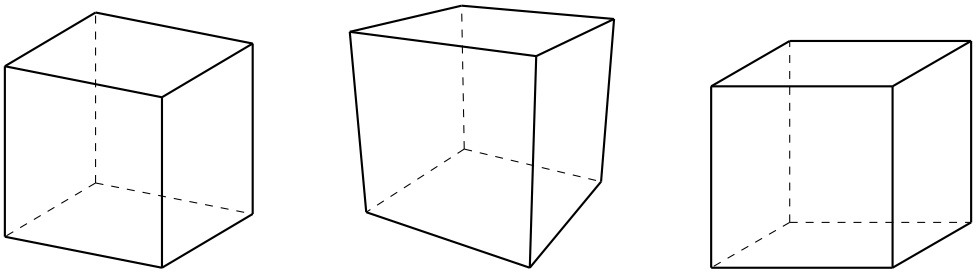
Camera	Projection
`camera{azimuth = a, elevation = e}`	orthographic (default): parallel projection, orthogonal to the page; the axis (Oz) stays vertical
`camera{..., perspective = true, distance = d}`	perspective (central projection): the eye is at distance d from the target
`camera{cavalier = true, angle = a, ratio = k}`	cavalier perspective: the plane (yOz) in true size, the receding axis x at the angle a with the ratio k

Other options: `target` (point aimed at, default the origin) and `scale`. The azimuth turns the figure around the vertical axis, the elevation shows more or less of the top.

Methods: `cam:project(P)` returns the 2D point (class `point`); `cam:to_z{A = z3.A, B = z3.B}` stores the projections in `z`.

```
\directlua{
  init_elements()
  function view(cam)
    init_elements()
    local S = scene3(cam)
    local CU = S:add(cube(point3(0, 0, 0), 2.4))
    S:polyhedron("cube", CU)
    S:export()
  end
}
\directlua{view(camera{azimuth = 30, elevation = 20})}
\begin{tikzpicture}
  \tkzDrawVisible[thick]{cube} \tkzDrawHidden{cube}
\end{tikzpicture}\hspace{1cm}
\directlua{view(camera{azimuth = 30, elevation = 20,
                       perspective = true, distance = 7})}
\begin{tikzpicture}
```

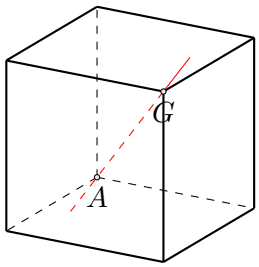
```
  \tkzDrawVisible[thick]{cube} \tkzDrawHidden{cube}
\end{tikzpicture}\hspace{1cm}
\directlua{view(camera{cavalier = true, angle = 30, ratio = 0.5})}
\begin{tikzpicture}
  \tkzDrawVisible[thick]{cube} \tkzDrawHidden{cube}
\end{tikzpicture}
```



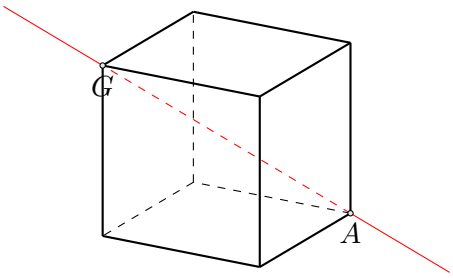
4.1 Changing the point of view

The same figure (the cube and the line (MN) of the first example) seen from four points of view. Only the camera changes: the visible and hidden parts are computed again for each view.

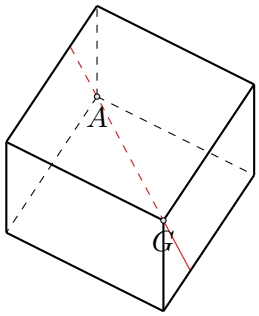
```
\directlua{
  init_elements()
  function cube_view(az, el)
    init_elements()
    local cam = camera{azimuth = az, elevation = el}
    local S = scene3(cam)
    local CU = S:add(cube(point3(0, 0, 0), 2.4))
    S:polyhedron("cube", CU)
    local A, G = CU.vertices[1], CU.vertices[7]
    S:segment("diag", A - 0.4 * (G - A), G + 0.4 * (G - A))
    S:export()
    cam:to_z{A = A, G = G}
  end
}
\def\View#1#2{%
  \directlua{cube_view(#1, #2)}%
  \begin{tikzpicture}[baseline]
    \tkzGetNodes
    \tkzDrawVisible[thick]{cube}  \tkzDrawHidden{cube}
    \tkzDrawVisible[red]{diag}    \tkzDrawHidden[red, dashed]{diag}
    \tkzDrawPoints(A,G)
    \tkzLabelPoints[below](A,G)
    \node[font = \small] at (0, -2.4) {azimuth #1, elevation #2};
  \end{tikzpicture}}
\View{30}{20}\hfill\View{120}{20}\par\bigskip
\View{30}{60}\hfill\View{-60}{-20}
```



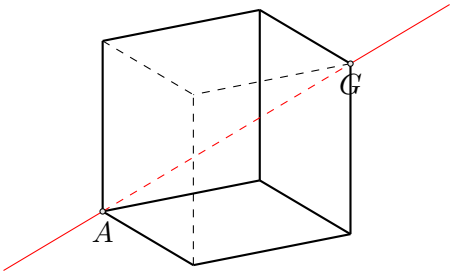
azimuth 30, elevation 20



azimuth 120, elevation 20



azimuth 30, elevation 60



azimuth -60, elevation -20

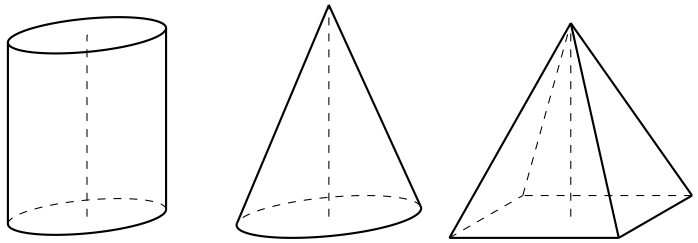
With a negative elevation the cube is seen from below; with an azimuth of 120° another face is in front, and the vertex A is no longer the farthest one, so the visible part of (MN) changes.

5 Objects

Constructor	Object
`line3(A, B), plane(A, B, C)`	line, plane (or `plane(P, normal)`)
`polyhedron(V, F)`	convex polyhedron: list of vertices, list of faces (lists of indices)
`cube(O, a)`	cube of center O, edge a
`tetrahedron(A, B, C, D)`	tetrahedron
`pyramid({A, B, ...}, S)`	pyramid of apex S (attributes `apex`, `base_center`)
`prism({A, B, ...}, v)`	prism (translation vector v)
`regular_polygon3(c, n, r, k)`	list of the k vertices of a regular polygon
`sphere(O, r)`	sphere
`ellipsoid(O, a, b, c)`	ellipsoid of semi-axes a, b, c (along x, y, z, or along a given basis)
`cylinder(B, T, r)`	cylinder (bases of centers B and T)
`cone(B, S, r)`	cone (base of center B, apex S)
`polygon3({A, B, ...}, opts)`	plane convex face (`color`, `fill`)
`open_box(O, L, W, H, opts)`	open box with flaps (`flaps = {a1, a2, a3, a4}`)

Every solid has the attribute `opaque`, `true` by default; it is given as the last argument, for example `cube(O, 4, false)` for a wireframe cube. Polyhedra have the attribute `centroid`.

```
\directlua{
  init_elements()
  local cam = camera{cavalier = true, angle = 30, ratio = 0.5}
  local S = scene3(cam)
  local CY = S:add(cylinder(point3(0, 0, 0), point3(0, 0, 3), 1.2))
  S:cylinder("cyl", CY)
  S:segment("axis", point3(0, 0, 0), point3(0, 0, 3))
  local CO = S:add(cone(point3(0, 4, 0), point3(0, 4, 3.5), 1.4))
  S:cone("cone", CO)
  S:segment("axis", point3(0, 4, 0), point3(0, 4, 3.5))
  local PY = S:add(pyramid({point3(1.4, 6.6, 0), point3(1.4, 9.4, 0),
    point3(-1.4, 9.4, 0), point3(-1.4, 6.6, 0)}, point3(0, 8, 3.2)))
  S:polyhedron("pyr", PY)
  S:segment("axis", PY.base_center, PY.apex)
  S:export()
}
\begin{tikzpicture}[scale = .8]
  \tkzDrawVisible[thick]{cyl}  \tkzDrawHidden{cyl}
  \tkzDrawVisible[thick]{cone} \tkzDrawHidden{cone}
  \tkzDrawVisible[thick]{pyr}  \tkzDrawHidden{pyr}
  \tkzDrawVisible{axis}        \tkzDrawHidden{axis}
\end{tikzpicture}
```



6 The scene: visible and hidden parts

A scene is created with a camera: `S = scene3(cam)`. The objects that can hide the others are declared with `S:add(obj)`; only the opaque ones hide something. Then each call below adds pieces to a *group*.

Method	Pieces added to the group
`S:segment(g, A, B)`	a segment
`S:circle(g, C, n, r)`	a circle (center, normal, radius)
`S:curve(g, f, opts)`	a curve $f(t)$, $t \in [0,1]$ (`closed`, `n`, `nohide`)
`S:polyhedron(g, P)`	the edges of a polyhedron
`S:sphere(g, SP)`	the apparent contour of a sphere
`S:ellipsoid(g, E)`	the apparent contour of an ellipsoid
`S:cylinder(g, CY)`, `S:cone(g, CO)`	rims and contour generators
`S:polygon_edges(g, L)`	the edges of a list of plane faces
`S:graph(g, G)`	the edges and vertices of a graph
`S:export()`	creates the paths and the counts

Three levels of hidden parts are handled:

1. *a polyhedron alone*: a face is seen if its outward normal points toward the camera; an edge is hidden if its two faces are hidden;
2. *curves on a sphere, contours of cylinders and cones*: the change between visible and hidden is at the points of contact with the apparent contour;
3. *several objects*: each line is cut where it crosses the contour of another object; each piece is tested by a ray toward the camera.

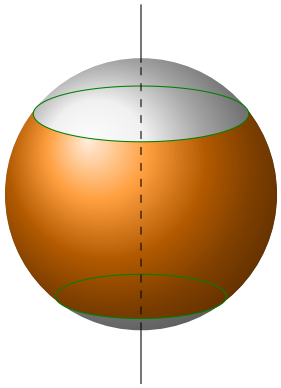
By convention, a segment inside an opaque solid (height, axis, radius of the base) is hidden.

7 Drawing

Macro	Drawing
`\tkzDrawVisible[opt]{g}`	visible pieces of the group `g`
`\tkzDrawHidden[opt]{g}`	hidden pieces (`dashed` by default)
`\tkzFillFaces[opt]{g}`	faces, from the farthest to the nearest, shaded by orientation
`\tkzFillRegion[opt]{r}`	fills a region
`\tkzClipRegion{r}`	clips to a region
`\tkzDrawBalls[color][r]{g}`	shaded balls at the vertices of a graph

Regions are closed paths built by `S:faces(g, L)` (faces), `S:sphere_zone(name, SP, axis, h1, h2)` (visible part of a zone of a sphere), `S:hull_region(name, points)` (outline of a convex surface) `S:ellipsoid_region(name, E)` (outline of an ellipsoid) and `S:curve_region(name, f)` (region bounded by a closed curve). Shadings, transparency and patterns are those of TikZ.

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 20, elevation = 15}
  local S = scene3(cam)
  z3.O = point3(0, 0, 0)
  local SP = S:add(sphere(z3.O, 1.8))
  local k = point3(0, 0, 1)
  S:sphere_zone("zone", SP, k, 1.1, -1.4)
  S:circle("circles", point3(0, 0, 1.1), k, math.sqrt(1.8^2 - 1.1^2))
  S:circle("circles", point3(0, 0, -1.4), k, math.sqrt(1.8^2 - 1.4^2))
  S:segment("axis", point3(0, 0, -2.6), point3(0, 0, 2.6))
  S:export()
  cam:to_z{O = z3.O}
}
\begin{tikzpicture}
  \tkzGetNodes
  \shade[ball color = gray!15] (O) circle (1.8);
  \begin{scope}
    \tkzClipRegion{zone}
    \shade[ball color = orange] (O) circle (1.8);
  \end{scope}
  \tkzDrawVisible[green!50!black]{circles}
  \tkzDrawHidden[green!50!black]{circles}
  \tkzDrawVisible{axis} \tkzDrawHidden{axis}
\end{tikzpicture}
```



8 The ellipsoid

`ellipsoid(O, a, b, c)` is the ellipsoid of center O and semi-axes a, b, c along the axes x, y, z:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1.$$

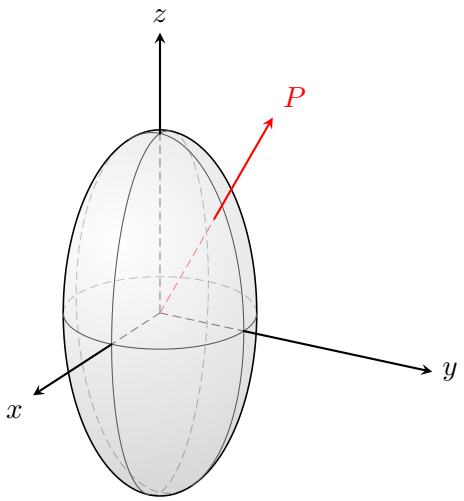
An inclined ellipsoid is given by an orthonormal basis of its axes: `ellipsoid(O, a, b, c, true, {e1, e2, e3})`. The affine map $(x, y, z) \mapsto (x/a, y/b, z/c)$ sends it to the unit sphere: the hidden parts and the apparent contour are computed there, so the contour is exact (it is an ellipse, with the three cameras).

Method	Result
`E:point(th, ph)`	point of longitude θ and latitude φ
`E:meridian(th)`	the meridian θ, a curve $f(t)$, $t \in [0, 1]$
`E:parallel(ph)`	the parallel φ, a curve $f(t)$
`E:contour(cam)`	the apparent contour, a curve $f(t)$
`S:ellipsoid(g, E)`	the contour in the group `g`
`S:ellipsoid_region(name, E)`	the outline as a region (to fill or clip)

The meridians and the parallels are drawn with `S:curve`: the ellipsoid, declared with `S:add`, hides their back part.

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 30, elevation = 22}
  local S = scene3(cam)
  z3.O = point3(0, 0, 0)
  local E = S:add(ellipsoid(z3.O, 1.6, 1.6, 3.2))
  S:ellipsoid("outline", E)
  S:ellipsoid_region("outline", E)
  S:curve("equator", E:parallel(0), {closed = true})
  S:curve("meridians", E:meridian(0), {closed = true})
  S:curve("meridians", E:meridian(math.pi / 2), {closed = true})
  z3.x, z3.y, z3.z = point3(4.2, 0, 0), point3(0, 5.2, 0), point3(0, 0, 5)
  z3.P = point3(0.6, 2.5, 4.2)
  S:segment("axes", z3.O, z3.x)
  S:segment("axes", z3.O, z3.y)
  S:segment("axes", z3.O, z3.z)
```

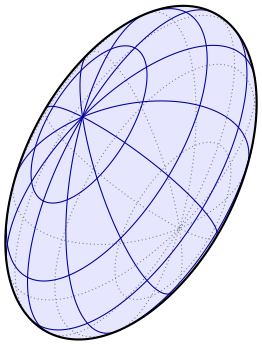
```
  S:segment("vecP", z3.O, z3.P)
  S:export()
  cam:to_z(z3)
}
\begin{tikzpicture}[scale = .8, line cap = round]
  \tkzGetNodes
  \tkzDrawHidden[densely dashed]{axes}
  \tkzDrawHidden[red, densely dashed]{vecP}
  \tkzDrawHidden[gray, densely dashed]{equator}
  \tkzDrawHidden[gray, densely dashed]{meridians}
  \begin{scope}
    \tkzClipRegion{outline}
    \shade[inner color = white, outer color = gray!60, opacity = .55]
      ([shift = {(-.7, 1.2)}]O) circle (4.5);
  \end{scope}
  \tkzDrawVisible[semithick]{outline}
  \tkzDrawVisible[gray!70!black]{equator}
  \tkzDrawVisible[gray!70!black]{meridians}
  \tkzDrawVisible[thick, -stealth]{axes}
  \tkzDrawVisible[red, thick, -stealth]{vecP}
  \tkzLabelPoint[below left](x){$x$}
  \tkzLabelPoint[right](y){$y$}
  \tkzLabelPoint[above](z){$z$}
  \tkzLabelPoint[above right, red](P){$P$}
\end{tikzpicture}
```



An inclined ellipsoid with its meridians and parallels:

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 20, elevation = 15}
  local S = scene3(cam)
  local R = rotation3(point3(0, 0, 0), point3(1, 1, 0), math.rad(40))
  local E = S:add(ellipsoid(point3(0, 0, 0), 3, 1.8, 1.1, true,
    {R(point3(1, 0, 0)), R(point3(0, 1, 0)), R(point3(0, 0, 1))}))
  S:ellipsoid("outline", E)
  S:ellipsoid_region("outline", E)
  for k = -2, 2 do
```

```
    S:curve("mesh", E:parallel(k * math.pi / 6), {closed = true})
  end
  for k = 0, 5 do
    S:curve("mesh", E:meridian(k * math.pi / 6), {closed = true})
  end
  S:export()
}
\begin{tikzpicture}
  \tkzFillRegion[blue!10]{outline}
  \tkzDrawHidden[gray, densely dotted]{mesh}
  \tkzDrawVisible[blue!60!black]{mesh}
  \tkzDrawVisible[thick]{outline}
\end{tikzpicture}
```



9 Transformations and graphs

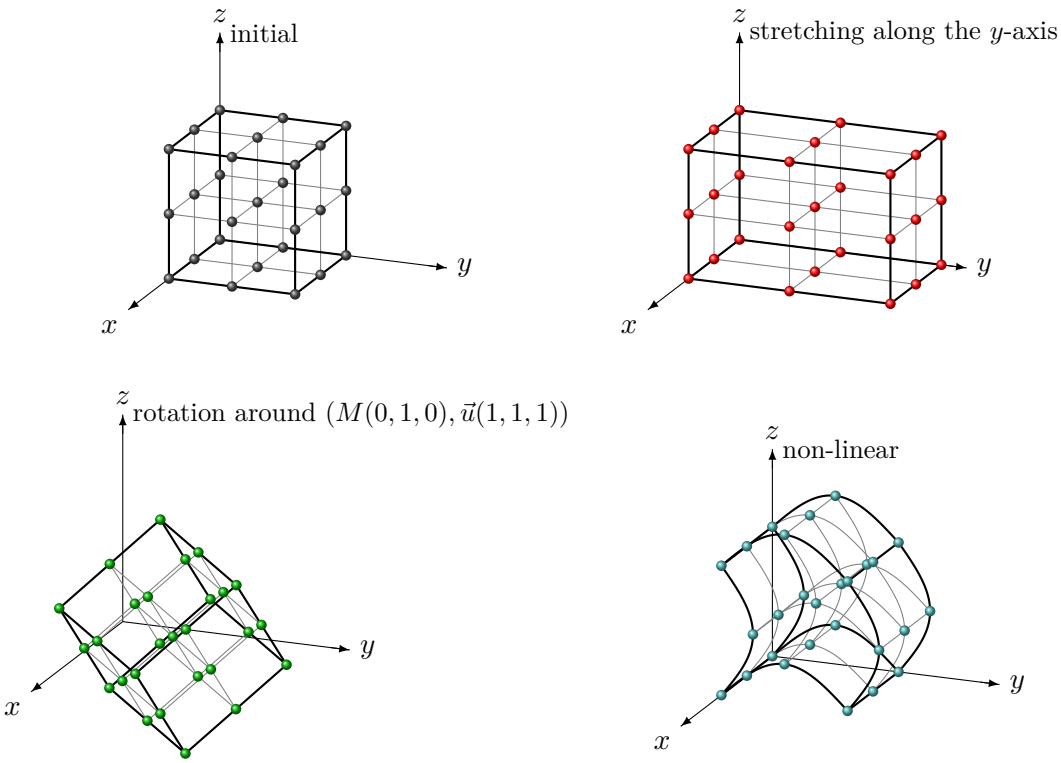
A transformation of space is a Lua function from `point3` to `point3`: `translation3(v)`, `scale3(kx, ky, kz [, c])`, `rotation3(c, u, angle)` (rotation around the axis $(c, \vec{u})$, angle in radians), `compose3(f, g)`, or any function.

A graph in space (`graph3`) has named vertices (`G:vertex(name, P)`) and edges (`G:edge(a, b, tag)`); `grid3(nx, ny, nz)` builds a grid. `G:map(f)` is the image of the graph by `f`; the edges are transformed too, so a non-linear map bends them. The vertices are nodes: they can be used by `tkz-euclide` or, later, by `tkz-graph`.

```
\directlua{
  init_elements()
  cam = camera{azimuth = 22, elevation = 18}
  G0 = grid3(3, 3, 3, 1)
  function panel(G)
    init_elements()
    local S = scene3(cam)
    S:graph("g", G)
    S:segment("axes", point3(0, 0, 0), point3(3.6, 0, 0))
    S:segment("axes", point3(0, 0, 0), point3(0, 3.6, 0))
    S:segment("axes", point3(0, 0, 0), point3(0, 0, 3.2))
    S:export()
    cam:to_z{x = point3(3.6, 0, 0), y = point3(0, 3.6, 0), z = point3(0, 0, 3.2),
             xa = point3(3.3, 0, 0), ya = point3(0, 3.3, 0), za = point3(0, 0, 2.9)}
  end
}
```

```
\def\Panel#1#2{%
  \begin{tikzpicture}[scale = .9]
    \tkzGetNodes
    \tkzDrawVisible[thin]{axes}
    \draw[-latex] (xa) -- (x);  \draw[-latex] (ya) -- (y);  \draw[-latex] (za) -- (z);
    \tkzDrawVisible[gray, thin]{ginner}
    \tkzDrawVisible[thick]{gframe}
    \tkzDrawBalls[#1]{g}
    \tkzLabelPoint[below left](x){$x$}
    \tkzLabelPoint[right](y){$y$}
    \tkzLabelPoint[above](z){$z$}
    \node[right, font = \small] at (z) {#2};
  \end{tikzpicture}}

\begin{tabular}{cc}
\directlua{panel(G0)}\Panel{black!70}{initial}
&
\directlua{panel(G0:map(scale3(1, 1.6, 1)))}\Panel{red}{stretching along the $y$-axis}
\\[1em]
\directlua{panel(G0:map(rotation3(point3(0, 1, 0), point3(1, 1, 1), math.rad(60))))}%
\Panel{green!70!black}{rotation around $(M(0,1,0), \vec{u}(1,1,1))$}
&
\directlua{panel(G0:map(function(p)
  return point3(p.x, p.y + 0.5 * math.sin(math.pi * p.z / 2),
   p.z + 0.6 * math.sin(math.pi * p.y / 2)) end))}%
\Panel{cyan!40!gray}{non-linear}
\end{tabular}
```



10 Space geometry in the classroom

This section shows figures of the last years of French secondary school (*terminale*): sections of a cube, intersections of planes, plane and sphere, constructions in a tetrahedron. They use the following tools.

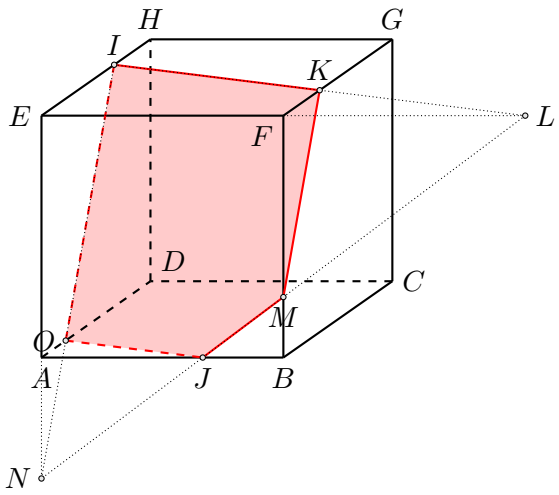
Syntax	Result
`named_cube(a [, O])`	cube $ABCDEFGH$ with the names of the textbooks (`CU.V.A` … `CU.V.H`); front face $ABFE$
`L:project(P), L:distance(P)`	projection on a line, distance
`L:intersect_line(L2)`	point of two coplanar lines (`nil` otherwise)
`L:intersect_plane(P)`	point of a line and a plane
`P:project(M), P:distance(M)`	orthogonal projection on a plane, signed distance
`P:intersect_plane(Q)`	line of intersection of two planes
`P:intersect_sphere(S)`	center and radius of the circle of intersection
`P:patch(a, b, c, angle, opts)`	rectangle of the plane, to draw it
`CU:section(P)`	section of a convex polyhedron (ordered points)
`F:split(P)`	the two parts of a face on each side of a plane
`S:polygon(g, pts)`	closed polygonal line (visible/hidden)
`S:plain(g, A, B)`	construction segment, no hidden part

10.1 Section of a cube by the plane (IJK)

The plane (IJK) cuts the cube along a pentagon. `CU:section(P)` gives it directly; the construction of the textbook is also drawn: $L = (IK) \cap (EF)$ in the plane of the top face, $M = (LJ) \cap (BF)$ and $N = (LJ) \cap (AE)$ in the plane of the front face, $O = (NI) \cap (AD)$ in the plane of the left face. The sides of the section on hidden faces are dashed automatically.

```
\directlua{
  init_elements()
  local cam = camera{cavalier = true, angle = 35, ratio = 0.55}
  local S = scene3(cam)
  local CU = S:add(named_cube(4))
  local V = CU.V
  for k, p in pairs(V) do z3[k] = p end
  z3.I = V.E + (2/3) * (V.H - V.E)
  z3.K = V.F + (1/3) * (V.G - V.F)
  z3.J = V.A + (2/3) * (V.B - V.A)
  local P = plane(z3.I, z3.J, z3.K)
  % construction
  z3.L = line3(z3.I, z3.K):intersect_line(line3(V.E, V.F))
  z3.M = line3(z3.L, z3.J):intersect_line(line3(V.B, V.F))
  z3.N = line3(z3.L, z3.J):intersect_line(line3(V.A, V.E))
  z3.O = line3(z3.N, z3.I):intersect_line(line3(V.A, V.D))
  % the section (computed directly)
  local sec = CU:section(P)
  S:faces("secfill", {polygon3(sec, {fill = "red!25, fill opacity = .8"})})
  S:polygon("secedges", sec)
  S:polyhedron("cube", CU)
  S:plain("cons", z3.I, z3.L)  S:plain("cons", z3.L, z3.N)  S:plain("cons", z3.N, z3.I)
  S:plain("cons", V.F, z3.L)   S:plain("cons", V.A, z3.N)
  S:export()
  cam:to_z(z3)
```

```
}
\begin{tikzpicture}[scale = .8]
  \tkzGetNodes
  \tkzFillFaces{secfill}
  \tkzDrawVisible[thick]{cube}       \tkzDrawHidden[thick, dashed]{cube}
  \tkzDrawVisible[red, thick]{secedges} \tkzDrawHidden[red, thick, dashed]{secedges}
  \tkzDrawVisible[densely dotted]{cons}
  \tkzDrawPoints(I,J,K,L,M,N,O)
  \tkzLabelPoints(A,B,J,M)
  \tkzLabelPoints[left](E,N,O)
  \tkzLabelPoints[above](H,G,I,K)
  \tkzLabelPoints[right](C,L)
  \tkzLabelPoints[above right](D)
  \tkzLabelPoints[below left](F)
\end{tikzpicture}
```

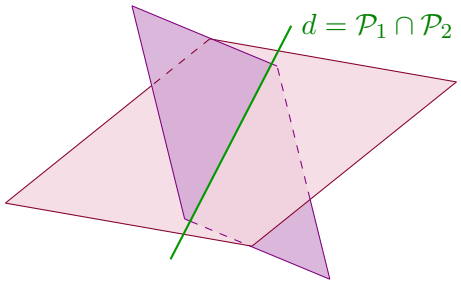


10.2 Two secant planes and their line of intersection

Two planes are drawn as rectangles (`P:patch`). Each one is cut along the other plane (`F:split`) so that the four pieces can be filled from the farthest to the nearest; the edges hidden by the other plane are dashed. The line of intersection is given by `P1:intersect_plane(P2)`.

```
\directlua{
  init_elements()
  local cam = camera{azimuth = -65, elevation = 22}
  local S = scene3(cam)
  local P1 = plane(point3(0, 0, 0), point3(0, 0, 1))
  local P2 = plane(point3(0, 0, 0), point3(1, 0.25, 0.45))
  local F1 = P1:patch(3.2, 1.8, nil, 0, {fill = "purple!15, fill opacity = .85"})
  local F2 = P2:patch(1.5, 2.6, nil, 90, {fill = "violet!35, fill opacity = .8"})
  S:add(F1) S:add(F2)
  local parts = F1:split(P2)
  for _, f in ipairs(F2:split(P1)) do table.insert(parts, f) end
  S:faces("planes", parts)
  S:polygon_edges("edges1", {F1})
  S:polygon_edges("edges2", {F2})
  local d = P1:intersect_plane(P2)
```

```
  local u = d:dir()
  S:segment("d", d.pa - 4.2 * u, d.pa + 4.2 * u)
  S:export()
  cam:to_z{lab = d.pa + 4.2 * u}
}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzFillFaces{planes}
  \tkzDrawVisible[purple!70!black]{edges1} \tkzDrawHidden[purple!70!black, dashed]{edges1}
  \tkzDrawVisible[violet]{edges2}          \tkzDrawHidden[violet, dashed]{edges2}
  \tkzDrawVisible[green!60!black, thick]{d} \tkzDrawHidden[green!60!black, thick, dashed]{d}
  \tkzLabelPoint[right, green!50!black](lab){$d = \mathcal{P}_1\cap\mathcal{P}_2$}
\end{tikzpicture}
```

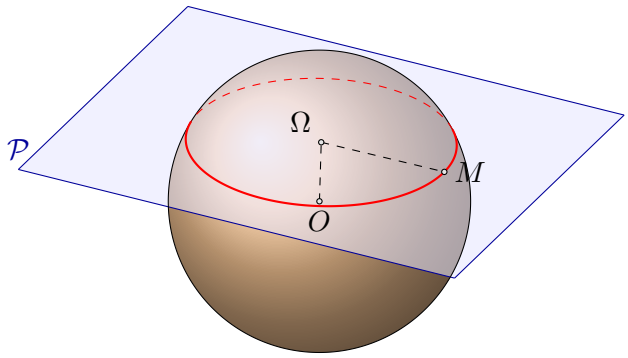


10.3 A plane and a sphere: the circle of intersection

The center Ω of the circle of intersection is the orthogonal projection of O on the plane, and its radius is $\sqrt{R^2 - d^2}$: `P:intersect_sphere(SP)` returns both. The plane is not opaque (it does not hide the sphere) but the sphere hides the back of the circle.

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 25, elevation = 18}
  local S = scene3(cam)
  z3.O = point3(0, 0, 0)
  local SP = S:add(sphere(z3.O, 2))
  local P = plane(point3(0, 0, 0.9), point3(0.15, 0.1, 1))
  z3.W, r = P:intersect_sphere(SP)
  local nrm = P.normal
  local e1 = (point3(0, 1, 0) - nrm:dot(point3(0, 1, 0)) * nrm):unit()
  z3.M = z3.W + r * e1
  local F = P:patch(3.2, 2.6, z3.W, 0, {fill = "blue!10, fill opacity = .55", opaque = false})
  S:faces("plane", {F})
  S:polygon_edges("pedges", {F})
  S:sphere("contour", SP)
  S:circle("section", z3.W, nrm, r)
  S:segment("radius", z3.O, z3.W)
  S:segment("radius", z3.W, z3.M)
  S:export()
  cam:to_z(z3)
  cam:to_z{lab = F.pts[3]}
}
\begin{tikzpicture}
  \tkzGetNodes
```

```
  \shade[ball color = orange!40] (O) circle (2);
  \tkzFillFaces{plane}
  \tkzDrawVisible[blue!60!black]{pedges} \tkzDrawHidden[blue!60!black, dashed]{pedges}
  \tkzDrawVisible{contour}
  \tkzDrawVisible[red, thick]{section}  \tkzDrawHidden[red, dashed]{section}
  \tkzDrawVisible{radius}                \tkzDrawHidden[dashed]{radius}
  \tkzDrawPoints(O,W,M)
  \tkzLabelPoint[below](O){$O$}
  \tkzLabelPoint[above left](W){$\Omega$}
  \tkzLabelPoint[right](M){$M$}
  \tkzLabelPoint[above, blue!60!black](lab){$\mathcal{P}$}
\end{tikzpicture}
```

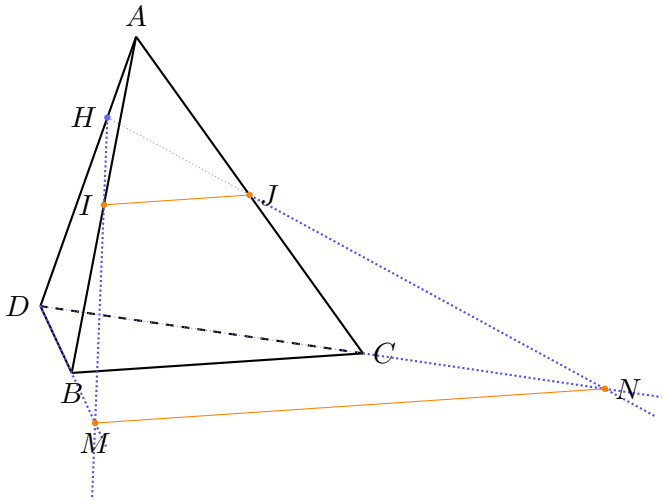


10.4 Roof theorem in a tetrahedron

I and J are the midpoints of $[AB]$ and $[AC]$, H a point of $[AD]$. The lines (HI) and (DB) are coplanar and meet at M, (HJ) and (DC) meet at N (`line3:intersect_line`). By the roof theorem, (MN) is parallel to (IJ) and (BC).

```
\directlua{
  init_elements()
  local cam = camera{azimuth = 15, elevation = 12}
  local S = scene3(cam)
  z3.A = point3(0.6, 0.6, 4.2)
  z3.B = point3(2.4, 0.2, 0)
  z3.C = point3(0.2, 3.6, 0)
  z3.D = point3(-1.6, -1.3, 0)
  local TE = S:add(tetrahedron(z3.A, z3.B, z3.C, z3.D))
  z3.I = midpoint3(z3.A, z3.B)
  z3.J = midpoint3(z3.A, z3.C)
  z3.H = z3.A + 0.3 * (z3.D - z3.A)
  z3.M = line3(z3.H, z3.I):intersect_line(line3(z3.D, z3.B))
  z3.N = line3(z3.H, z3.J):intersect_line(line3(z3.D, z3.C))
  S:polyhedron("tetra", TE)
  local function con(a, b, k) S:segment("cons", a, b + k * (b - a)) end
  con(z3.H, z3.M, 0.25)  con(z3.H, z3.N, 0.1)  con(z3.D, z3.M, 0.2)  con(z3.D, z3.N, 0.1)
  S:segment("par", z3.I, z3.J)
  S:segment("par", z3.M, z3.N)
  S:export()
  cam:to_z(z3)
}
\begin{tikzpicture}
```

```
  \tkzGetNodes
  \tkzDrawVisible[thick]{tetra}  \tkzDrawHidden[thick, dashed]{tetra}
  \tkzDrawVisible[blue!70, densely dotted, thick]{cons}
  \tkzDrawHidden[blue!45, densely dotted]{cons}
  \tkzDrawVisible[orange]{par}   \tkzDrawHidden[orange, dashed]{par}
  \tkzDrawPoints[orange](I,J,M,N)
  \tkzDrawPoints[blue!60](H)
  \tkzLabelPoints[above](A)
  \tkzLabelPoints(B,M)
  \tkzLabelPoints[left](D,H,I)
  \tkzLabelPoints[right](C,J,N)
\end{tikzpicture}
```

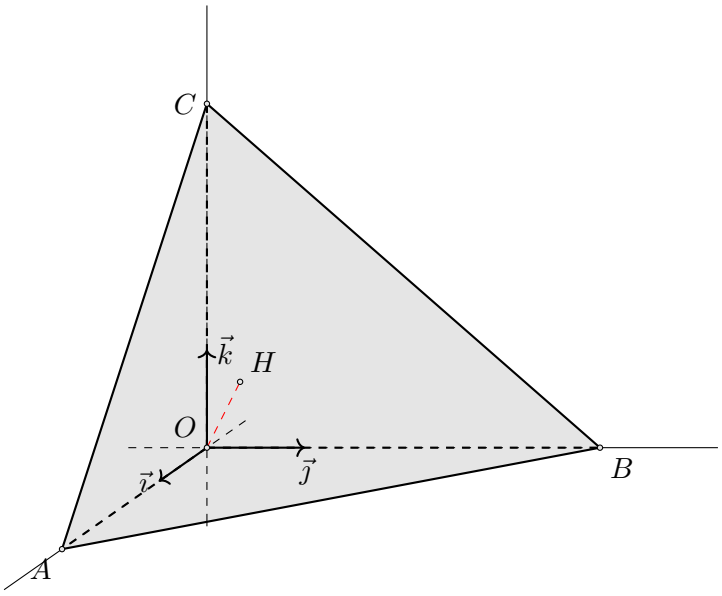


10.5 Distance from a point to a plane: O and the plane (ABC)

H is the orthogonal projection of O on the plane (ABC) (`P:project(z3.O)`); OH is the distance from O to the plane. The segment $[OH]$, inside the tetrahedron, is dashed.

```
\directlua{
  init_elements()
  local cam = camera{cavalier = true, angle = 35, ratio = 0.6}
  local S = scene3(cam)
  z3.O = point3(0, 0, 0)
  z3.A = point3(3, 0, 0)
  z3.B = point3(0, 4, 0)
  z3.C = point3(0, 0, 3.5)
  local P = plane(z3.A, z3.B, z3.C)
  z3.H = P:project(z3.O)
  local T = polygon3({z3.A, z3.B, z3.C}, {fill = "gray!25, fill opacity = .8", opaque = false})
  local TE = S:add(tetrahedron(z3.O, z3.A, z3.B, z3.C))
  S:faces("face", {T})
  S:polyhedron("tetra", TE)
  S:segment("OH", z3.O, z3.H)
  S:segment("axes", point3(-0.8, 0, 0), point3(4.2, 0, 0))
  S:segment("axes", point3(0, -0.8, 0), point3(0, 5.2, 0))
  S:segment("axes", point3(0, 0, -0.8), point3(0, 0, 4.5))
  S:export()
```

```
  cam:to_z(z3)
  cam:to_z{i = point3(1, 0, 0), j = point3(0, 1, 0), k = point3(0, 0, 1)}
}
\begin{tikzpicture}[scale = 1.3]
  \tkzGetNodes
  \tkzFillFaces{face}
  \tkzDrawVisible{axes}            \tkzDrawHidden[dashed]{axes}
  \tkzDrawVisible[thick]{tetra}    \tkzDrawHidden[thick, dashed]{tetra}
  \tkzDrawVisible[red]{OH}         \tkzDrawHidden[red, dashed]{OH}
  \tkzDrawSegments[->, thick](O,i O,j O,k)
  \tkzDrawPoints(O,A,B,C,H)
  \tkzLabelPoints[below left](A)
  \tkzLabelPoint[above left](O){$O$}
  \tkzLabelPoints[below right](B)
  \tkzLabelPoints[left](C)
  \tkzLabelPoints[above right](H)
  \tkzLabelPoint[left](i){$\vec\imath$}
  \tkzLabelPoint[below](j){$\vec\jmath$}
  \tkzLabelPoint[right](k){$\vec k$}
\end{tikzpicture}
```



11 Gallery

More examples (cube and inscribed sphere, perspective views, open cartons, zone of a sphere, mesh of an ellipsoid, paraboloid and volume element, transformations of a grid…), with their code, are published on altermundus.fr.

12 Limits of the present version

- Polyhedra must be convex.
- The lines are cut by sampling and bisection (200 steps for a segment, 360 for a circle); an exact computation of the crossings is planned.

- Cylinders and cones are replaced by inscribed prisms and pyramids with 256 faces for the tests of hidden parts (the drawing uses the exact circles and generators).
- The outline of a surface of revolution is computed as a convex hull: it is exact only for convex surfaces.
- Faces are filled by the painter's algorithm (sorting by depth), which can fail for faces that cross each other.